An HLA Compliant Agent-Based Fast-Time Simulation Architecture For Analysis Of Civil Aviation Concepts

Jesse Aronson[†]*, Vikram Manikonda*[*]*, Wilbur Peng*[*]*, Renato Levy*[*] *and Karlin Roth*[‡]

[†]Science Applications International Corporation
1100 N. Glebe Road, Suite 1100, Arlington, VA 22201
aronsonj@saic.com

[*]Intelligent Automation Incorporated
7519 Standish Place, Suite 200, Rockville, MD 20855
{vikram, wpeng, rlevy}@i-a-i.com

[‡]NASA Ames Research Center
Moffett Field, CA 94035-1000
kroth@mail.arc.nasa.gov

Keywords:
Agents, OpenCybele, HLA, Composable Simulations, Air Traffic Modeling and Control

ABSTRACT*: This paper describes the architecture of the Airspace Concept Evaluation System (ACES), a fast-time simulation being developed by NASA as an analysis tool for evaluating novel concepts in air traffic management. ACES has the goal of supporting a wide range of studies and evaluations. Researchers will use ACES for evaluating a wide range of air traffic management concepts and technologies. Consequently the ACES architecture is required to be computationally efficient, flexible and modular. At the core of the ACES architecture is the High Level Architecture (HLA) RTI integrated with Cybele, an agent infrastructure. The paper describes how the agent- and HLA-based approach chosen for ACES satisfies customer requirements, how an ACES federation operates and provides an operational concept of how researchers will work with the system.*

1. Introduction

Over time, the demand for air travel continues to increase. With an average traffic growth rate of over 4% per year current studies predict significant increases in delays and air traffic congestion in the coming years. [1] There is significant concern that the current NAS operational paradigm cannot accommodate the forecasted steadily increasing air travel demand. NASA, as a leader in Air Traffic Management research, has made improvement of the National Airspace System one of its highest priorities. In 2002 NASA initiated the Virtual Airspace Modeling and Simulation (VAMS) Project, a five year research and development effort in response to the projected growth of the demand for air travel and lack of sufficient system capacity to meet this demand. The need to evaluate the costs and benefits of new operational paradigms early in the development process is of paramount importance in identifying the most promising concepts. In recognition of this critical evaluation function, one objective of the NASA VAMS Project is to develop a national simulation and modeling capability for system-level design and tradeoff studies. This objective is being met through the development of the Airspace Concept Evaluation System (ACES) fast-time simulation [1][3]. This paper describes the architecture of the ACES system and shows how this architecture meets VAMS project requirements.

The initial major application of ACES will be in evaluating a set of operational concepts being developed within VAMS. These operational concepts range from insertion of technologies into particular segments of the NAS (e.g., the use of automation to increase runway throughput at airports) to system-wide operational changes (e.g., shifting traffic away from “hub and spoke” operations towards “point to point” service). Some

concepts are complementary and may ultimately be combined, while others offer competing solutions to particular NAS capacity challenges. Evaluations of operational concepts must be able to include cost, capacity and safety metrics.

The need to flexibly evaluate a varied set of individual and combined concepts leads to a set of simulation system requirements including: flexible simulations that can be tailored to individual researchers' needs ("plug and play modeling toolkit"); standardized modeling interfaces to allow for easy integration of new models and legacy simulations; ability to scale to multiple processors to achieve required fast-time performance; ability to integrate legacy codes; and data collection that is easily adaptable to the researcher's specific needs.

This paper describes the architecture of the of the ACES system. At the core of the system is the High Level Architecture (HLA) integrated with Cybele™ [4] an agent-based modeling and simulation framework. The agent infrastructure provides the modeling and simulation framework at the federate level while the HLA RTI provides the infrastructure at the inter-federate level. The architecture fosters composability by (i) insulating models from the specifics of particular simulation frameworks, thus making it easier to reuse them in a range of environments; (ii) allowing models to be composed both within the federate and at the FOM level; and (c) providing core services such as time management, event management, thread management and communication services at the federate level that are compatible with the RTI. The architecture reduces the effort required on the part of the modeler by abstracting away the details of distributed simulation.

The paper is organized as follows. In Section 2 we define agents and their relation ship to federate and federations. In Section 3 we present the design of the ACES architectures. Specific details of the agent simulation framework and its interaction with the RTI Are defined in sections 4 and 5 respectively. A brief overview of the model design and development from a modelers' perspective is presented in Section 6. Simulation Management and data collection are discussed in Section 7. Section 8 provides conclusions and directions for future work.

2. Terminology and Definitions

A software agent is often defined as persistent software entity that acts autonomously on behalf of a user by receiving (via sensors) inputs (messages, events) and acts (sends a response, changes an internal state) in response to the inputs (See [5][6] and references therein). Attributes of agents can also include intelligence, fault tolerance, mobility, proactive behavior, adaptability etc.

From a software perspective we define an *agent* as a software object that possesses the following conceptual characteristics:

- *Encapsulation of local state*. Agents encapsulate state and cannot directly access the state of other agents. This enables agents to be reconfigured and distributed across nodes.
- *Independent execution*. Agents independently control how and when they execute. There is no single execution control structure that controls agent execution.
- *Message and event driven behavior*. Agents interact with the world and with other agents by communicating through some well defined-set of messages and protocols.

From a modeling and simulation perspective we view an agent as having the following organization that is based on the Activity Centric Programming Paradigm (ACP) and characteristics [7]:

- An agent defines an autonomous module of a particular simulation that interacts with other agents via messages/events. Agents are also typically the smallest unit of the simulation that can be distributed. Example: Flight Agent, Center Air Traffic Control (ATC) Agent, Airport Agent.
- Each agent is composed of *activities* where each activity encapsulates a particular role/behavior of the agent. Example: ATC agents can be decomposed into activities corresponding to Conflict Detections & Resolution (CD&R), aircraft vectoring and inter-controller communication and handoffs.
- Each activity is further composed of (i) a *role layer* that describes the syntax and semantics of the different types of interactions that an agent can perform in an application. Roles capture the formalities of an interaction-protocol. For example a role layer can define the interaction protocol of handoff between controllers (ii) a *domain layer* that represents the domain expertise (e.g., flight dynamics, reasoning component, sorting etc) of the agent and can be implemented in any software/languages or could even be legacy software; and (iii) a *glue layer* consisting of a set of adapter objects that implement the interface required by a role by making appropriate method invocations on the local domain objects. The adaptor layer loosens the coupling between models and communications protocols, facilitating reuse of models across multiple applications.

In a simulation we view the relationship between agents, federates and federations as follows

- A federate is composed of a set of individual agents. For example a North East federate could be composed on the set of airports, air traffic control facilities and flights in the northeast corridor.
- The federation describes the combined system that is composed of constituent independently executing programs known as federates.

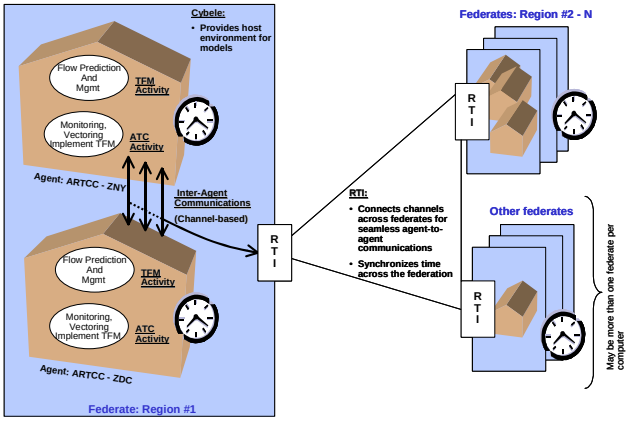

Figure 1: Activity/Agent/Federate/Federation Hierarchy

Figure 1 depicts the relationship between agents and their activities, federates and federations in ACES.

3. ACES Architecture

3.1 Architectural Requirements

ACES is being developed to satisfy a set of goals and requirements encompassing both runtime and development-time lifecycle phases. These include:

Run-time: ACES is a fast-time distributed simulation system. At runtime a collection of networked computers work together to generate a representation of the NAS, with each computer being assigned some subset of a scenario's domain elements and support functions (e.g., data collection). The primary goal of using multiple computers is to obtain higher performance via parallel execution. Consequently, the primary job of the run-time architecture is to provide and integrated execution environment that allows the federation's computers to run in a coordinated and efficient fashion. This is accomplished by providing a distributed time management facility that ensures that the various federates stay synchronized, an efficient data distribution mechanism that works within time management to ensure that messaging between federates adheres to the causality of the simulation (that is, messages always get delivered to the application in order of simulation time, regardless of network delivery order) and a set of simulation control functions that allow the federates to perform coordinated start-up, shut-down, etc.

Development-time: ACES also has the driving requirement of being configurable, based on a *model toolbox*. Composability of ACES executions as needed for particular studies is required both within federates and between federates. The first level of composability is intra-federate, model-level composition. This level deals with models within federates. Ideally, this level of composition should not only facilitate model development and federate assembly but also insulate the model developer from needing to consider the vagaries of distributed simulation.

*Interoperability***:** The second level of ACES composition exists between federates. ACES complies with the High Level Architecture. Thus, entire federates, whether or not they are built using the ACES infrastructure, can be executed together as long as they are HLA-compliant and use the ACES FOM and Federation Agreements.

*Usability***:** The simulation system must not only execute quickly but also be usable by analysts. Run-time usability features include the ability to control the simulation from a single computer and the ability to support visualization of scenario evolution. Usability also includes capabilities to connect the simulation to the analytical context in which it is used. This includes the ability to collect data from the simulation and manipulate simulation data into formats which can be fed to downstream assessment tools.

Life-cycle Features: Last, the use and management of analytical simulation tools can be greatly aided by lifecycle tools that manage the model toolbox and system databases, create scenarios and automate linkages to external data. The ACES system design considers such capabilities. Such tools are, however built "around" the simulation and so are not included in ACES system requirements, detailed architecture or planned capabilities.

3.2 Architectural Vision

Figure 2 shows the overall system architecture, which can be seen as a set of applications that share a common infrastructure. The infrastructure supports both run-time communications, focused on high-performance data exchange and coordination for simulations, and non-runtime communications, focused on managing persistent data (simulation inputs and outputs). As shown in Figure 2, the simulation infrastructure is built on a foundation of standard networking protocols provided by the computer operating system (shown in green). Layered above this are industry-standard inter-process communications

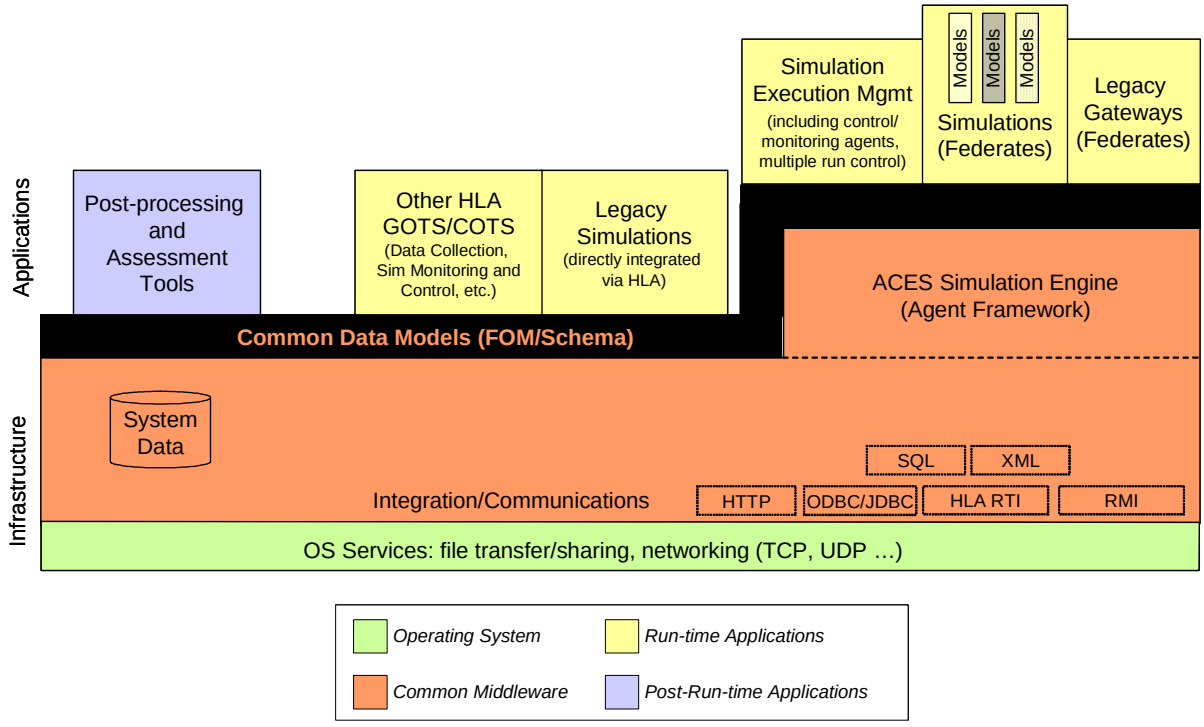

Figure 2: Unified System Architecture

protocols including HTTP, Remote Method Invocation and ODBC/JDBC for database access. This layer of the architecture also includes the HLA RTI, which provides the backbone of run-time communication. Non-runtime communication is based primarily upon ODBC/JDBC database communications.

The communications layer includes several standards for representing data and commands. Data representation standards include the HLA Federation Object Model as expressed by the HLA Object Model Template as well as the Extensible Markup Language (XML) promulgated by the World Wide Web Consortium (W3C).

Above these infrastructure components sits the ACES simulation engine/agent framework. ACES federates are not required to use this framework; as shown in Figure 2, COTS, GOTS and legacy codes can be used in ACES federations. Use of the simulation engine, however greatly facilitates development of simulations made up of "plug-and-play" models by: enforcing clean model interfaces, providing modelers a layer of insulation from distributed communications and time coordination details and providing built-in basic services for simulation time and event management.

ACES infrastructure components and applications are connected not only through common infrastructure functionality but also by common data models. Common data models take the form of Federation Object Models (which are mapped with matching inter-agent message classes) and database schemata. These common models are the foundation of interoperability among run-time applications as well as between run-time and post-processing tools. Common schemata could also be used to add simulation initialization data into this common picture.

Last are the ACES applications themselves. ACES run-time applications include Cybele-based simulation federates, utility and control federates implementing functions such as control of simulation rate, monitoring and data collection. The various run-time applications together create an integrated synthetic representation of the National Airspace System. The applications execute as independent programs across multiple computers, using the infrastructure to communicate, coordinate and store data. Post-runtime applications can access the stored data to support analyses of experimental data collected from simulation executions.

4. Agent Simulation Infrastructure for Federates

As discussed earlier ACES uses an RTI compliant agent infrastructure and agent-based approach to model and simulate entities within a federate.

The agent infrastructure is based on IAI's OpenCybele™/Cybele™ agent infrastructure. Cybele is

built on the top of the Java 2 platform and provides the runtime environment for control and execution of agents. The architecture consists of a kernel and several service implementations.

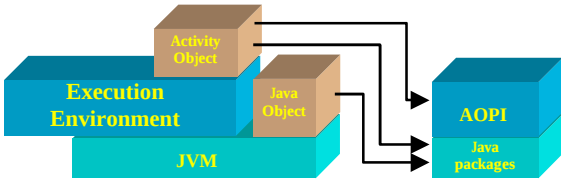

Figure 3; The layers of Activity objects and its execution environment

The architecture kernel provides certain application interface methods for agent programmers to write classes representing activities using the ACP paradigm. These methods are called Activity Oriented Programming Interface (AOPI) methods. The AOPI's in turn use the published interfaces to different agent services. The execution environment supporting ACP is a layer above Java Virtual Machine. (See Figure 3). Since ACP is a restricted version of OOP, the execution environment supporting ACP restricts the functionality of activity objects to be event-driven. Note that the activity classes also use the java packages in addition to AOPI.

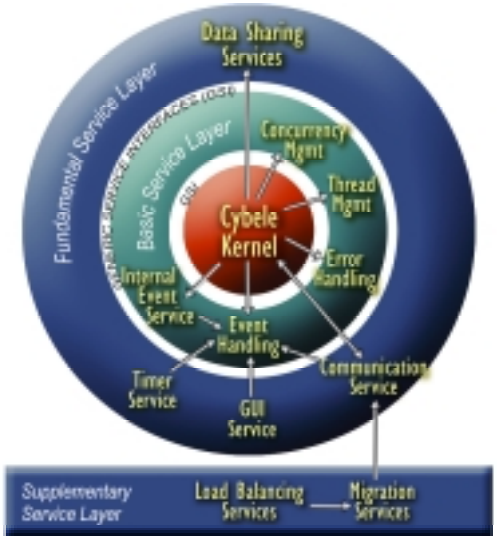

Figure 4: Service Layered Architecture

The agent-infrastructure adopts a service-layered architecture promoting plug-n-play capability of agent services (see Figure 4). The services and their interfaces are defined in such a way that performance can be fine-tuned by loading different service implementations as appropriate to the OS/platform/network and/or the agent application domain, without having to re-write the agent code

The agent services are categorized into three types of layers–-basic, fundamental, and supplemental--based on their relevance to a typical agent runtime environment. The basic are the essential services required for creating agents, activities, agent operation, and inter-activity communication. They include error handling, concurrency management, event handling, thread-management, and internal event services. The fundamental services are primarily event generation services that are essential for autonomous and communicative multi-agent system. They allow an agent to send messages to another agent that is either co-located on the same host or is on a host that is across the network. They also allow an agent to set an alarm and generate a self timer-event. Services included in the fundamental layer are communication, timer, data sharing and GUI services. The supplemental services are services that enhance performance and include migration and load balance services

The architecture supports the (ACP) paradigm, distributed computing/simulation, location independent channel based publish-subscribe messaging including, synchronous, asynchronous, broadcast and point-to-point messaging.

5. Agent Infrastructure-HLA Interactions

Integrating the Cybele infrastructure with the RTI involved several steps. First, the native Cybele inter-agent communications service had to be replaced with HLA-compliant communications. Second, the Cybele timer, event management and thread management service were adapted to support time-managed, fast-time discrete event simulation and sender-side filtering. In this section we discuss the details of interfaces of the communication, timer and event management services with the HLA/RTI.

5.1 Interfacing with the RTI: Interest Management

The agent infrastructure uses a concept of *message channels* for communications. Agents publish messages to particular channels using the provided AOPI's, and other agents can opt to receive messages by subscribing to channels. The RTI uses a similar mechanism, however in place of channels the RTI uses the more generic notion of *routing spaces*. If routing spaces are not defined then interactions are forwarded to all other federates subscribed to that class of interaction; routing spaces optimize network utilization by allowing data to be

filtered at the sending end so that interactions get send only to the federates interested in that interaction.

Conversion between Cybele communications notions (serialized Java objects over message channels) and RTI notions (FOM objects over RTI Data Management or Data Distribution Management) involves several steps which are done, transparent to the user, by the infrastructure.

First, each federate knows the interests of the other federate on the network. If no other federate is interested in a published message then the sending federate delivers it to its local channels but it is not distributed by the RTI. This feature is known as Sender Side Filtering.

If a remote interest in the data has been established, then the infrastructure checks to see if there is a matching FOM interaction for this Cybele message object. If the FOM interaction does not exist then the object is packaged as a serialized Java object within a special FOM interaction class. This interaction is forwarded to the other federates, which reconstitute the object into the appropriate Java class and pass it to the appropriate Cybele channel.

If, however, a matching FOM class is found, the attributes of the Java class are copied into the matching attributes of the FOM class (matching requires identical names between Cybele and FOM attribute names). Any "leftover" parameters in the Cybele object are streamed into an extra interaction attribute called "reserved". At the receiving end an object of the appropriate Java class is again reconstituted from the data in the FOM interaction and passed to the appropriate Cybele channel.

The integrated agent-HLA approach eliminates the need for the modeler to distinguish between inter-and intra-federate messaging. Agents simply send and receive messages and the common agent infrastructure takes care of dispatching messages to other interested agents, whether they are local or in different federates. This approach allows simulation developers to define a message once; inter-agent (intra-federate) messages are automatically carried through to the inter-federate level.

This approach also provides automation in FOM agility. The contents of inter-agent messages can be augmented without affecting the FOM, as each FOM class contains a holder for packed "extra" attributes and the infrastructure automates translation between native Java and FOM formats. The infrastructure also minimizes required maintenance when both the Java message classes and the FOM change, as the hookup between the two message representations is automated.

Last, the ACES infrastructure is able to handle objects with greater complexity than can natively be represented in a FOM. ACES message classes can contain sub-objects as well as collections of both primitive and object types. The infrastructure handles such objects by converting their complex components into XML strings. These strings are passed as RTI attributes and are reconstituted at the receiving end.

5.2 Interfacing with the RTI Time Management: Timer Service

Time management and synchronization is critical in any distributed simulation. To enable composability it is critical that the models and/or simulations issues related to time management across the simulation be decoupled from the models. To ensure this the infrastructure timer service is built around the concept of *logical time*. (The logical time is also referred to as simulation time since it represents the current time within the simulation. The time management approach uses HLA's event driven next-event request (NER) functionality to advance time.

Each Cybele-HLA federate has associated with it the *current logical time t*, that is initially initialized during start-up across the federation by a global discrete clock. The models use the various timer AOPI's to set timers/alarms, with respect to the local logical time, to trigger/process events at desired intervals.

Models are written as event-driven agents that are *independent* of the underlying time management infrastructure. They do not need to know anything about advancing or managing time. Rather, they need only to respond to individual events and how to schedule events using timers. For example, a flight model can have a timer that sends a message regarding its position and speed to a "controller" agent every 10 logical time units.

Each federate is in one of two states, the *message processing state* or *the advancing time state*. In the message processing state, the federate is in the process of dispatching all messages at the current logical time *t*. As agents process messages and timer events, new messages timestamped at *t+1* may be generated, or new timer events may be scheduled at future logical times. Once all messages on the system message queue have been processed, the federate seeks to advance time to the next logical time at which it has events to process by entering the advancing state

In the advancing time state, the federate may continue to receive messages timestamped at *t+1*. However, it is guaranteed *not* to generate any more messages since all messages have been processed.

5.3 ACES Event Management

The ACES event management service works in conjunction with the time management service to insure that messages are delivered in a predictable and deterministic manner. It is designed to support *repeatable* simulation under any configuration. This means that given the same set of agents with the same initial conditions, the system will produce the exact same results under any assignment of agents to Cybele-HLA federates. This is made possible by the fact that the agent is the basic unit of distribution: each agent can be modeled as a deterministic entity that takes events in a stream and produces new events in the "future."

In order to support repeatability, it is important to ensure that all events always arrive in the same order. For example if agents A, B, and C are sending to agent X from different federates, the order of messages arriving at logical time *t* is not guaranteed to be the same, since HLA RTI does not order messages with the same timestamp. Moreover, internal messages and timer events must also be ordered correctly.

To insure deterministic message deliver, each message is tagged with the sending agent, a sequence number and the activity identity. During the beginning of each message processing state, all of the events for each agent are sorted according to a specific comparison criterion that produces deterministic results.

Although the Cybele-HLA enables repeatable, reconfigurable simulation, individual models/agents must still be written in a manner that allows deterministic behavior which is independent of local configuration. For example, it is necessary to ensure that all agents that use pseudorandom number sequences have their own seeds and generators and do not share them with other agents.

6. Developing Models in ACES

From a models perspective developing/coding models in ACES reduces to coding agents and their interactions with other agents using the AOPI's corresponding to creating agents, creating activities, setting up timers, publishing and subscribing to appropriate channels, and invoking appropriate callback in response to events (messages and timers). From a modelers perspective no modification to the agent software needs to be made to compose a set of agents into a federation, join the federation and interact with other simulations via the RTI. These activities are handled by the agent infrastructure. Transparent to the user the infrastructure (i) maps Cybele channels into appropriate RTI routing space for inter-federate communications; (ii) translates intra-federate serialized messages into FOM complaint interactions when they go across the network; (iii) implements message transport; and (iv) Integrates Cybele-time management with RTI time management.

7. Simulation Control and Data Collection

As an HLA federation the ACES system is amenable to standard federation control functionality. The primary simulation control module is the Visualization/Scenario Tool (VST). The VST, which is itself a Cybele-based federate, performs a number of functions. First, it controls initialization for the entire ACES federation. In ACES each Cybele-based federate starts up in a generic mode – it does not immediately create any model agents. The VST allocates functionality to the various federates, providing information to each as to which domain entities it is assigned and consequently what set of agents it is to instantiate. The VST also provides runtime visibility into the federation execution. It includes a plan view display and can also display a range of flight and air traffic control parameters. The VST and related helper applications also include a range of other utility functions including speed control (allowing the simulation to be slowed down or coordinated with wall-clock time), synchronized start-up, and distribution of initialization data.

Data collection in the present build of ACES is accomplished primarily through the GOTS DCT data collection tool. This has the advantage of being configurable and was able to be implemented with a modest effort, however limits data collection to FOM data. Future builds will investigate mechanisms for collecting federates' internal state data as well as FOM information.

8. Conclusions

The ACES design provides a unique approach to the modeling and simulation of new NAS concepts. By acknowledging the need to provide a flexible and dynamic simulation capability, the ACES design utilizes the HLA / Agent Infrastructure to realize these critical requirements. The Modeling Toolkit approach, and the agent-based paradigm used to structure the models and their interactions, provides a modeling structure that correlates to the NAS operational environment.

Current architectural work focuses on enhancing the multiple run/Monte Carlo simulation features, performance optimization, implementation of a richer

data collection mechanism and continued support for population of the model toolkit.

The combination of these new capabilities with the proven agent-based infrastructure will provide a simulation tool capable of supporting complex, distributed fast-time simulation to meet NASA's research requirements.

9. Acknowledgements

ACES development effort is part of a three and a half year project supported by the Air Traffic Management System Development and Integration Contract team consisting of The Raytheon Co., Seagull Technology, SAIC, Intelligent Automation, Inc. (IAI), TITAN, Metron, Booz-Allen Hamilton and others. The authors wish to acknowledge the efforts of the many individuals in these organizations that have contributed to the design and development of the ACES prototype. Specifically, we would like to acknowledge technical contributions by Pauline Froemberg, Mary Ellen Miller and Ed Stevens of Raytheon; Marlin Johnson, Roger Wuerfel and Thomas Lee from SAIC; Anna Teittinen, and David Ditzenberger from IAI; Paul Abramson of PDA Associates (TITAN, Inc); Doug Sweet, and George Hunter of Seagull Technology.

10. References

[1] Federal Aviation Administration, Aviation Policy and Plans (APO), http://api.hq.faa.gov/pubs.asp.

[2] Doug Sweet, Vikram Manikonda, Jesse Aronson, and Karlin Roth. Fast-Time NAS Simulation for Analysis of Advanced ATM concepts. In AIAA Modeling and Simulation Conference and Exhibit, 5 - 8 Aug 2002 Monterey, California

[3] See http://www.asc.nasa.gov/vams/tim2/download/09_Roth(NRT).pdf for details on ACES

[4] OpenCybele,http://www.opencybele.org, Intelligent Automation Incorporated.

[5] Michael Wooldridge. Introduction to Multiagent Systems, John Wiley and Sons, 2002

[6] Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence by Gerhard Weiss (Editor), Cambridge, MA, The MIT Press, 1999

[7] Kutluhan Erol, Jun Lang and Renato Levy. Designing Agents from Reusable Components. In proceedings of the Fourth International Conference on Autonomous Agents, Barcelona, Spain 2000.

Author Biographies

JESSE ARONSON is a Senior Technical Manager at Science Applications International Corporation. He holds a M.S. from Polytechnic Institute of New York and a B.E. from The Cooper Union, both in Electrical Engineering. He is a licensed Professional Engineer. Mr. Aronson has been active in the government simulation community for fourteen years. Prior to this Mr. Aronson designed sonar and satellite navigation systems.

VIKRAM MANIKONDA is the Director of the Distributed Intelligent Systems Group at Intelligent Automation Incorporated. He received his B.E. degree in Electrical Engineering from the Birla Institute of Technology and Science, India, in 1992, his M.S. and Ph.D. degrees, both in Electrical Engineering, from the University of Maryland at College Park, in 1994 and 1997 respectively. His research interests include intelligent control, robotics and multi-agent systems, modeling and simulation

WILBUR PENG is a Research Scientist at Intelligent Automation, Inc. He obtained a B.S. from Cornell University and a PhD from the University of Maryland in electrical engineering. His research interests currently include intelligent agents, machine intelligence, pattern recognition, and neural networks.

RENATO LEVY is a Principal Research Scientist at Intelligent Automation Incorporated. He received his B.S. degree in Electrical Engineering from the Federal University of Rio de Janeiro, Brazil in 1986, his MBA degree in Administration and Finance from the Institute for Business and market economy (IBMEC), Brazil. Mr. Levy is currently a Ph.D. candidate at George Washington University specializing in the field of Computer Science with a major in Distributed and Parallel Computing

KARLIN ROTH is the Chief of the Aerospace Operations Modeling Office within the Aviation Systems Division at NASA Ames Research Center. She has responsibility for technical and programmatic development of research leading to a system-level evaluation capability for the airspace system. Dr. Roth earned PhD and MS degrees in Aerospace Engineering from the University of Florida and a BS degree in Applied Mathematics from the Indiana University of Pennsylvania.